Amendments to the Claims:

This listing of claims replaces all prior versions and listings of claims in the application:

Listing of Claims:

1. (Original) In a computer program, a method for maintaining dependencies among a set of objects each having a value, the set of objects including an object A and an object B, the method for maintaining dependencies comprising:

when the value of object A is a function of the value of object B and the value of object B changes, marking object A as dirty and not recomputing the value of object A until object A is queried for a value;

when the value of object B changes, invalidating the dependents of object B and all of their further dependents, including severing dependencies among the dependents of object B and all of their further dependents; and

causing each invalidated observer-only object to recompute its value by querying the values of the objects from which the observer-only object depends.

2. (Original) The method of claim 1, further comprising:

providing object B in the construction of object A, wherein the value of object A is a function of the value of the object B that was provided in the construction of object A.

3. (Currently amended) The method of claim 1, further comprising:

providing in object B a ~~handleRequest~~ method that adds a requester owned by object A to a dependents list for object B, the dependents list identifying all objects whose value is a function of the value of object B.

4. (Previously presented) The method of claim 3, wherein the dependents lists for all objects in the set of objects collectively define a directed, acyclic dependency graph.

5. (Original) The method of claim 1, further comprising:

when an object is marked as dirty, breaking any dependency relationships the marked object may have had; and

when the value of an object is recomputed, identifying the objects on which the recomputed value is actually dependent and identifying the recomputed object as dependent only on the identified objects.

6. (Currently amended) The method of claim 1, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid ~~(i.e., dirty)~~.

7. (Previously presented) In a computer program, a method comprising:

identifying the objects upon which a given object depends as those objects into which the given object passed itself as a requester during execution of a compute method of the given object;

marking the given object as dirty whenever the value of any one of the identified objects changes and not recomputing the value of the given object until the given object is queried for a value; and

modifying a dependency graph by severing dependencies among dependents of the identified objects with changed value and all of their further dependents to create a modified dependency graph.

8. (Original) The method of claim 7, further comprising:

identifying as dependents of a root object all objects that passed themselves as requester objects to the root object or to a dependent of the root object during execution of the requester objects' respective compute methods, whereby the set of dependents of the root object is a set that changes based on the computation of dependents and not the root object itself.

9. (Currently amended) The method of claim 7, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid ~~(i.e., dirty)~~.

10. (Canceled)

11. (Currently amended) A method for changing objects having values defining state of a computer program application, comprising:

receiving a change to a value of a changed object, the changed object having objects depending directly on the changed object and objects depending indirectly on the changed object through an object different from the changed object, the changed object being a settable object in the computer program application;

registering the change with a transaction;

dirtying all objects dependent ~~(directly or indirectly)~~ on the changed object;

severing dependencies from the changed object and all of its direct and indirect dependent objects; and

whenever a leaf object is encountered as a dependent object, enqueuing the leaf object for synchronization after the transaction is committed.

12. (Canceled)

13. (Currently amended) The method of claim 11, wherein leaf object synchronization comprises:

recomputing a value for each object[[s]] marked as dirty, identifying the objects on which the recomputed value is actually dependent, and identifying the recomputed object as dependent only on the identified objects.

14. (Original) The method of claim 13, further comprising:

using a requester object to make the transaction consistent, the requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

15. (Previously presented) A method for changing objects defining state of a computer program application, comprising:

creating a transaction and registering with the transaction one or more changes to settable objects, each change being made to a corresponding changing object;

for each change registered, traversing a dependency graph from the changing object and (i) for each dependent object on the dependency graph, marking the dependent object as dirty and detaching the dependent object from the dependency graph, and (ii) accumulating each leaf object encountered in traversing the dependency graph in a strobe queue; and

traversing the strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph.

16. (Original) The method of claim 15, wherein:

the dependency graph represents application state;

the roots of the dependency graph are the settable objects of the application state; and

the intermediate nodes of the dependency graph are dependent objects whose values are the results of intermediate computations.

17. (Original) The method of claim 15, wherein:

the leaf objects of the dependency graph are coupled to a user interface.

18. (Original) The method of claim 17, wherein:

the leaf objects are coupled directly to the user interface.

19. (Previously presented) In a computer program, a method for managing dependency among a set of objects, each object of the set of objects having a value, the set of objects including dependent objects, wherein a given object can have objects depending directly on the given object and objects depending indirectly on the given object through other objects of the set of objects, each dependent object having a value that is a function of the values of one or more of the other objects in the set of objects, the method comprising:

calculating the dependency among objects in the set of objects dynamically at the time objects calculate their values.

20. (Previously presented) The method of claim 19, wherein each observed object in the set of objects has one or more accessor methods that each take a requester argument and return a current value of the observed object, the requester argument identifying the object requesting the value of the observed object.

21. (Currently amended) The method of claim 19, wherein each settable object in the set of objects has a value-setting method that takes two arguments;~~, namely~~ a transaction argument identifying a transaction with which the change to the settable object's value is registered and a new value for the settable object.

22-27. (Canceled)

28. (Original) A system for maintaining dependencies among a set of objects in a computer program, each object having a value, the set of objects including an object A and an object B, the system comprising:

means for recomputing the value of object A, wherein when the value of object A is a function of the value of object B and the value of object B changes, marking object A as dirty and not recomputing the value of object A until object A is queried for a value;

means for recomputing the value of object B, wherein when the value of object B changes, invalidating the dependents of object B and all of their further dependents, including severing dependencies among the dependents of object B and all of their further dependents; and

means for causing each invalidated observer-only object to recompute its value by querying the values of the objects from which the observer-only object depends.

29. (Currently amended) A system ~~for maintaining dependencies among a set of objects in a computer program, each object having a value, the system~~ comprising:

means for identifying the objects upon which a given object depends as those objects into which the given object passed itself as a requester during execution of a compute method of the given object; [[and]]

means for marking the given object as dirty whenever the value of any one of the identified objects changes and not recomputing the value of the given object until the given object is queried for a value[[.]]; and

means for modifying a dependency graph by severing dependencies among dependents of the identified objects with changed value and all of their further dependents to create a modified dependency graph.

30. (Canceled)

31. (Currently amended) A system for changing objects having values defining state of a computer program application, comprising:

means for receiving a change to a value of a changed object, the changed object having objects depending directly on the changed object and objects depending indirectly on the changed object through an object different from the changed object, the changed object being a settable object in the computer program application;

means for registering the change with a transaction;

means for dirtying all objects dependent ~~(directly or indirectly)~~ on the changed object; and

means for severing dependencies from the changed object and all of its direct and indirect dependent objects; wherein

whenever a leaf object is encountered as a dependent object, the leaf object is enqueued for synchronization after the transaction is committed.

32. (Previously presented) A system for changing objects defining state of a computer program application, comprising:

means for creating a transaction and registering with the transaction one or more changes to settable objects, each change being made to a corresponding changing object;

means for traversing a dependency graph, for each change registered, from the changing object and (i) for each dependent object on the dependency graph, marking the dependent object as dirty and detaching the dependent object from the dependency graph, and (ii) accumulating each leaf object encountered in traversing the dependency graph in a strobe queue; and

means for traversing the strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph.

33. (Previously presented) A system for managing dependency among a set of objects in a computer program, each object of the set of objects having a value, the set of objects including dependent objects, wherein a given object can have objects depending directly on the given object and objects depending indirectly on the given object through other objects of the set of objects, each dependent object having a value that is a function of the values of one or more of the other objects in the set of objects, the system comprising:

means for determining a time at which objects calculate their values; and

means for calculating the dependency among objects in the set of objects dynamically at the time objects calculate their values.

34. (Canceled)

35. (Original) A computer program product, tangibly stored on a computer-readable medium, for maintaining dependencies among a set of objects each having a value, the set of objects including an object A and an object B, the product comprising instructions operable to cause a computer to:

recompute the value of object A, wherein when the value of object A is a function of the value of object B and the value of object B changes, object A is marked as dirty and the value of object A is not recomputed until object A is queried for a value;

recompute the value of object B, wherein when the value of object B changes, the dependents of object B and all of their further dependents are invalidated, and the dependencies among the dependents of object B and all of their further dependents are severed; and

cause each invalidated observer-only object to recompute its value by querying the values of the objects from which the observer-only object depends.

36. (Currently amended) A computer program product, tangibly stored on a computer-readable medium, ~~for maintaining dependencies among a set of objects in a computer program, each object having a value,~~ the product comprising instructions operable to cause a computer to:

identify the objects upon which a given object depends as those objects into which the given object passed itself as a requester during execution of a compute method of the given object; [[and]]

mark the given object as dirty whenever the value of any one of the identified objects changes and not recompute the value of the given object until the given object is queried for a value[[.]] ; and

modify a dependency graph by severing dependencies among dependents of the identified objects with changed value and all of their further dependents to create a modified dependency graph.

37. (Canceled)

38. (Currently amended) A computer program product, tangibly stored on a computer-readable medium, for changing objects having values defining state of a computer program application, the product comprising instructions operable to cause a computer to:

receive a change to a value of a changed object, the changed object having objects depending directly on the changed object and objects depending indirectly on the changed object through an object different from the changed object, the changed object being a settable object in the computer program application;

register the change with a transaction;

dirty all objects dependent ~~(directly or indirectly)~~ on the changed object;

sever dependencies from the changed object and all of its direct and indirect dependent objects; and

whenever a leaf object is encountered as a dependent object, enqueue the leaf object for synchronization after the transaction is committed.

39. (Previously presented) A computer program product, tangibly stored on a computer-readable medium, for changing objects defining state of a computer program application, the product comprising instructions operable to cause a computer to:

create a transaction and registering with the transaction one or more changes to settable objects, each change being made to a corresponding changing object;

traverse a dependency graph, for each change registered, from the changing object and (i) for each dependent object on the dependency graph, marking the dependent object as dirty and detaching the dependent object from the dependency graph, and (ii) accumulating each leaf object encountered in traversing the dependency graph in a strobe queue; and

traverse the strobe queue after all changes to settable objects have been registered and synchronizing each leaf object by recomputing values for objects marked as dirty and rejoining recomputed objects with the dependency graph, whereby leaf objects are rejoined with the dependency graph.

40. (Previously presented) A computer program product, tangibly stored on a computer-readable medium, for managing dependency among a set of objects in a computer program, each object of the set of objects having a value, the set of objects including dependent objects, wherein a given object can have objects depending directly on the given object and objects depending indirectly on the given object through other objects of the set of objects, each dependent object having a value that is a function of the values of one or more of the other objects in the set of objects, the product comprising instructions operable to cause a computer to:

calculate the dependency among objects in the set of objects dynamically at the time objects calculate their values.

41. (Canceled)

42. (Previously presented) The product of claim 35, further comprising instructions operable to:

provide object B in the construction of object A, wherein the value of object A is a function of the value of the object B that was provided in the construction of object A.

43. (Currently amended) The product of claim 35, further comprising instructions operable to:

provide in object B a ~~handleRequest~~ method that adds a requester owned by object A to a dependents list for object B, the dependents list identifying all objects whose value is a function of the value of object B.

44. (Previously presented) The product of claim 43, wherein the dependents lists for all objects in the set collectively define a directed, acyclic dependency graph.

45. (Previously presented) The product of claim 35, further comprising instructions operable to:

break any dependency relationships the marked object may have had when an object is marked as dirty; and

when the value of an object is recomputed, identify the objects on which the recomputed value is actually dependent and identify the recomputed object as dependent only on the identified objects.

46. (Currently amended) The product of claim 35, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid ~~(i.e., dirty)~~.

47. (Previously presented) The product of claim 36, further comprising instructions operable to:

identify as dependents of a root object all objects that passed themselves as requester objects to the root object or to a dependent of the root object during execution of the requester objects' respective compute methods, whereby the set of dependents of the root object is a set that changes based on the computation of dependents and not the root object itself.

48. (Currently amended) The product of claim 36, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid ~~(i.e., dirty)~~.

49. (Currently amended) The product of claim 38, wherein leaf object synchronization comprises:

recomputing a value for each object[[s]] marked as dirty, identifying the objects on which the recomputed value is actually dependent, and identifying the recomputed object as dependent only on the identified objects.

50. (Currently amended) The product of claim 49 [[38]], further comprising instructions operable to:

use a requester object to make the transaction consistent, the requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

51. (Previously presented) The product of claim 39, wherein:

the dependency graph represents application state;

the roots of the dependency graph are the settable objects of the application state; and

the intermediate nodes of the dependency graph are dependent objects whose values are the results of intermediate computations.

52. (Previously presented) The product of claim 39, wherein:

the leaf objects of the dependency graph are coupled to a user interface.

53. (Previously presented) The product of claim 52, wherein:

the leaf objects are coupled directly to the user interface.

54. (Previously presented) The product of claim 40, wherein each observed object in the set has one or more accessor methods that each take a requester argument and return a current value of the observed object, the requester argument identifying the object requesting the value of the observed object.

55. (Currently amended) The product of claim 40, wherein each settable object in the set has a value-setting method that takes two arguments:~~, namely~~ a transaction argument identifying a transaction with which the change to the settable object's value is registered and a new value for the settable object.

56-57. (Canceled)

58. (Previously presented) The method of claim 19, wherein all objects of the set are instantiated from object-oriented programming classes that inherit a set of methods from a common base class.

59. (Currently amended) The method of claim 58, wherein the common base class ~~is a requester class with~~ has one or more methods to lock down and reset queried values in order to guarantee consistency.

60. (New) The product of claim 40, wherein all objects of the set are instantiated from object-oriented programming classes that inherit a set of methods from a common base class.

61. (New) The product of claim 60, wherein the common base class has one or more methods to lock down and reset queried values in order to guarantee consistency.

62. (New) The system of claim 28, further comprising:

means for providing object B in the construction of object A, wherein the value of object A is a function of the value of the object B that was provided in the construction of object A.

63. (New) The system of claim 28, further comprising:

means for providing in object B a method that adds a requester owned by object A to a dependents list for object B, the dependents list identifying all objects whose value is a function of the value of object B.

64. (New) The system of claim 63, wherein the dependents lists for all objects in the set collectively define a directed, acyclic dependency graph.

65. (New) The system of claim 28, further comprising:

means for breaking any dependency relationships the marked object may have had when an object is marked as dirty; and

when the value of an object is recomputed, means for identifying the objects on which the recomputed value is actually dependent and means for identifying the recomputed object as dependent only on the identified objects.

66. (New) The system of claim 28, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid.

67. (New) The system of claim 29, further comprising:

means for identifying as dependents of a root object all objects that passed themselves as requester objects to the root object or to a dependent of the root object during execution of the requester objects' respective compute methods, whereby the set of dependents of the root object is a set that changes based on the computation of dependents and not the root object itself.

68. (New) The system of claim 29, wherein the set of objects includes settable objects and dependent objects, and each dependent object maintains a flag whose setting marks the dependent object as valid or invalid.

69. (New) The system of claim 31, further comprising:

for leaf object synchronization, means for recomputing a value for each object marked as dirty, means for identifying the objects on which the recomputed value is actually dependent, and means for identifying the recomputed object as dependent only on the identified objects.

70. (New) The system of claim 69, further comprising:

means for using a requester object to make the transaction consistent, the requester object operating to request an object's value so that the requested value cannot change until the requester terminates, at which time all objects whose values were requested by the requester object are released.

71. (New) The system of claim 32, wherein:

the dependency graph represents application state;

the roots of the dependency graph are the settable objects of the application state; and

the intermediate nodes of the dependency graph are dependent objects whose values are the results of intermediate computations.

72. (New) The system of claim 32, wherein:

the leaf objects of the dependency graph are coupled to a user interface.

73. (New) The system of claim 72, wherein:

the leaf objects are coupled directly to the user interface.

74. (New) The system of claim 33, wherein each observed object in the set has one or more accessor methods that each take a requester argument and return a current value of the observed object, the requester argument identifying the object requesting the value of the observed object.

75. (New) The system of claim 33, wherein each settable object in the set has a value-setting method that takes two arguments: a transaction argument identifying a transaction with which the change to the settable object's value is registered and a new value for the settable object.

76. (New) The system of claim 33, wherein all objects of the set are instantiated from object-oriented programming classes that inherit a set of methods from a common base class.

77. (New) The system of claim 76, wherein the common base class has one or more methods to lock down and reset queried values in order to guarantee consistency.